基于中间盒的 SDN 网络信息服务中心策略实施架构 *

李海龙，张　钊，董思岐，胡　磊

(火箭军工程大学 保障学院，西安 710025)

摘　要：中间盒是一种网络管理员手动设置行为策略的设备。软件定义网络（software-defined network，SDN）的出现使得中间盒实施策略的可能性变得多样化。为改善信息服务中心的安全防护，提出一种无须管理员参与即可响应网络事件的基于 SDN 的动态中间策略实施架构，提出可以满足控制器与中间盒之间通信的接口。在虚拟机中实施了具有防火墙和入侵防御系统（intrusion prevention system，IPS）的中间盒原型来评估策略执行体系，验证原型获得的实验效果。结果表明，该体系结构能够在不影响网络性能的前提下动态执行中间盒策略，使网络应用程序能够正常运行。

关键词：中间盒；SDN；策略实施

中图分类号：TP393.07　　doi: 10.19734/j.issn.1001-3695.2018.06.0458

Middleware-based SDN network information service center policy implementation framework

Li Hailong, Zhang Zhao, Dong Siqi, Hu Lei

(*School of Military Operation Support*, *Rocket Force University of Engineering*, *Xi'an* 710025, *China*)

Abstract: A middlebox is a device that a network administrator manually sets behavior policies. The advent of Software-defined networks (SDN) has made it possible to diversify the possibilities of implementing a middlebox implementation strategy. In order to improve the security of the information service center, this paper proposes an SDN-based dynamic intermediate policy implementation architecture that can respond to network events without administrators' participation, and proposes an interface that can satisfy the communication between the controller and the intermediate box. An intermediate box prototype with a firewall and an intrusion prevention system (IPS) was implemented in the virtual machine to evaluate the policy execution system and verify the experimental results obtained by the prototype. The results show that the architecture can dynamically execute the middlebox policy without affecting network performance, so that the network application can run normally.

Key words: middlebox; SDN; policy enforcement

0　引言

信息服务中心的安全防护系统，用于确保信息服务基础设施安全可靠运行，信息资源安全受控访问，以及不同域间信息的安全交换，防范外部攻击和非授权访问。信息服务中心安全防护系统由信息服务中心局域网安全防护系统、信息服务保护系统和域间信息传送安全控制系统组成。信息服务中心局域网安全防护系统主要部署防火墙、入侵检测、防病毒和虚拟机安全防护等系统，并采用自主可控、可信计算等措施，提升系统安全防护能力，保证信息服务中心基础设施安全可靠运行。

传统网络配置中，安全系统的布控主要依赖于网络管理员的人为操作，往往是网络管理员针对不同的网络威胁在网络中作出相应的防御策略，但是这种人为的安防措施耗费精力较大，同时防御效率也相对较低，SDN 技术的可编程器件及其对网络视图的集中控制使得安全系统的自动化有了可行性。

1　相关研究

中间盒是在计算机网络中执行交换和路由之外功能的中间网络元件[1]。它们扮演着诸如安全增强（例如，侵入检测/预防系统和防火墙），性能改进（如 WAN 优化器和代理）和 IP 地址保护（网络地址转换器(network address translation,NAT）)等至关重要的角色。中间盒的行为通常由网络管理员手动和主动配置的策略来定义，网络管理员需要预测网络中可能发生的事件。这种做法比较麻烦，原因如下：

a）政策配置过程很容易出错，因为它很复杂，高度依赖于人为干预；

b）网络运营可能无法正确或及处理事件；

c）现在的许多应用程序都是动态的，并且建立了很难预测

收稿日期：2018-06-10；**修回日期**：2018-08-06　　**基金项目**：国家自然科学基金资助项目（61374054）

作者简介：李海龙（1978-），男，甘肃庆阳人，副教授，硕导，博士，主要研究方向为计算机网络（657455664@qq.com）；张钊（1993-），男，河北廊坊人，硕士，主要研究方向为计算机网络；董思岐（1995-），女，黑龙江肇东人，硕士，主要研究方向为计算机网络；胡磊（1995-），男，浙江金华人，硕士，主要研究方向为计算机网络．

的连接[2]。

在软件定义网络中，数据转发和控制层之间是分离的，一个或多个中央控制器参与网络中的规则配置。这让网络设备具有了更好的可编程性[3]。

由于控制功能的集中化和网络设备的可编程性，SDN 为处理中间盒及其配置的新方法提供了可能。在文献[4~8]中介绍了一些相关主题的方法。在文献[4,5]中，SDN 控制器使用南向接口来删除或添加策略。文献[4]通过设计 API 来实现联合控制，文献[5]提出的体系结构通过迁移中间盒的实例来实施策略，但是，以上两种方法在内部状态发生变化的时候都仍需要管理员对原有设计进行改进，要求管理员至少在一个实例上手动配置策略，所以上述两种个方法并没有摆脱认为控制的这一需求。文献[6]提出 FlowTags 架构，使用增强型中间架构修改中间盒的特定内部功能，增强控制器的可扩展性，从而达到降低数据包处理延迟的效果。在文献[7]中，建议动态地配置交换机以通过一系列中间盒（例如防火墙→代理→NAT）转发业务。由于中间盒产生的变化可能会导致数据包不能正确转发，所以使用 SDN 控制器来映射中间盒前后数据包的信息。与中间盒本身没有任何交互作用，这种映射只能通过控制器完成。上述的文献[6、7]都讨论了如何在一个中间盒上设置策略，但是可能干扰其他中间盒的正确策略设置，所以操作的方便性并不是十分的好。文献[8]通过改变 SDN 控制器来获取中间盒的修改内容，并且能够通知中间盒其他设备所做的改变。利用特定的南向接口来执行控制器和中间盒之间的通信。文献[8]提出的体系结构描述了源和目的地之间的最短路径包含所有必需的中间盒，但是 SDN 控制器和中间盒之间没有交互。 总之，上述的文献提出了一些处理使用 SDN 特性带来的数据包修改带来的问题的方法。但是，它们都没有消除在中间盒中手动配置策略这一流程，这是管理员要执行的一项复杂工作。而且，其中一些如[7]和[8]不考虑 SDN 控制器和中间盒之间的通信，这降低了它们的集成度，并限制了中央控制器的优势。

本文提出了一种中间盒策略执行体系结构，利用 SDN 中控制器集中实现的可编程接口，使中间盒能够处理动态网络应用程序，并且无须人工配置就可以响应网络中的事件。使用软件代理充当 SDN 控制器和中间盒之间的代理，同时能够使用 OpenFlow 协议与基础架构设备进行交互，而无须在 OpenFlow 中进行任何修改。

2　策略执行机制

本文提出的策略执行机制利用可编程器件和集中控制的优势缓解 SDN 网络中间盒配置的压力。它允许动态的中间盒策略执行，并且对这些网络中的常规接口进行最小限度的更改。所提出的架构，如图 1 所示，使用 SDN 控制器的集中视图，从中间盒获得的数据，以动态和自动的方式选择最合适的策略。

使用一个可选的软件代理来保障控制器和中间盒之间的交互，代理程序获取中间盒中已经存在的数据，并将该数据发送到控制器上运行的应用程序，以便与控制器所掌握的集中式流程信息一起进行分析。例如，代理可以获取中间存储的日志或配置文件，然后将这些数据发送给应用程序。代理对中间盒和控制器都是透明的。这种方式下，架构能确保与当前 SDN 标准的兼容性。中间盒的功能没有变化，网络中的控制器或其他设备不需要修改。

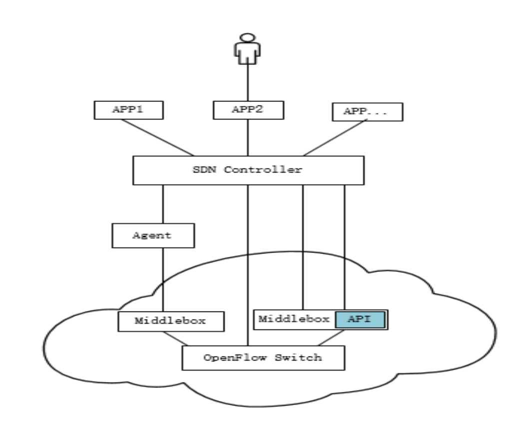


图 1　体系结构组成部分

Fig.1　Architecture component

另外，控制器完整的网络视图可以使策略得到优化。这种方案允许中间盒在应用时向控制器沟通，以便控制器可以使用该信息来配置其他中间盒或交换机。例如，IPS 识别恶意流量并将其通知给控制器，控制器可以将这些流量转发给另一个 IPS 进行更深入的检查，或者配置一个交换机来丢弃这个流量。

这些操作可以通过在控制器上运行的应用程序自动控制，或者发送给管理员进行进一步分析。这种方式下，网络管理员在配置中间盒方面仍然起着关键作用。但是管理员可以创建应用程序来自动执行此任务，一次将策略应用于多个中间盒，并非是依次配置每个中间盒。如果管理员事先授权策略中的更改措施，策略可能会根据网络中的事件自动更改，使中间盒更加灵活。

本文中进一步描述和评估的例子是在控制器上运行并使用中间软件代理配置防火墙的应用程序。此应用程序会分析控制器收到的数据包，以确定防火墙是允许还是拒绝具有特定特性的通信。本文使用 Python 下的防火墙（Iptables[9]）和 IPS（Snort[10]）软件代理，下文将详细地介绍所提出的架构的每个组件。

2.1　控制器和应用程序

所提出的架构是使用基于 Ryu 框架的控制器来实现的。Ryu 框架是 SDN 的一个主动更新的开源框架，支持最新版本的 OpenFlow 和其他各种协议，同时支持 OpenStack[11]。当控制器收到来自交换机的包输入消息时，它将该消息传递给正在运行的应用程序。这种类型的消息包含表征网络流量记录的信息，这些信息被提取并用于规划中间盒中设置的策略。

2.2　与中间盒的通信

专有的中间盒通常具有允许与外部代理交互的应用程序接口以及开发应用程序以扩展其功能。但是，由设备执行操作的控制受到制造商的限制，中间盒的修改受限。另一方面，开源

的中间盒可以根据开发者的需求进行修改。然而，这样的中间盒较为复杂，修改它们的代价也较昂贵。

鉴于上述原因，本文提出的体系结构提供了两种选择来提供控制器和中间盒之间的通信：a）通过在中间盒中添加 API 对其进行修改；b）对于中间盒是专有的或其修改过于复杂的情况，由中间代理人进行。

在这两个选项中，数据都是使用 JavaScript 对象表示法（JSON）格式进行交换，JSON 允许数据以紧凑的方式传输，使传输速度更快。策略按以下格式传输：

a)数据包标头,策略处理的流量信息（如 IP 地址、端口）。

b)状态,策略的状态（如允许或拒绝由防火墙阻止/释放执行的操作，或代理缓存的状态）。

添加策略到中间盒的请求示例如下：

{“context”:{scr_ip”:192.168.1.10”,
“dst_ip”:”168.168.1.11”,
“scr_port”:”54532”,
“dst_port”:”4345”,
“action”:{“action”:”block”}}

该接口也发出对控制器请求的响应。在成功的请求中，发送 HTTP 代码 200 来识别这个状态。在不成功的请求中，错误代码与描述问题的消息一起被传送。

当软件代理与控制器沟通的同时，倘若是通过增加 API 的形式，由于添加的 API 直接添加到了中间和当中，并且使用同样的数据传输格式，故而因此增加的传输开销相对较小；当使用代理人修改中间盒过于复杂当情况时，在此种情况下，由于中间盒需要作出当修改过于复杂，所以尤其自身进行修改调试的代价要远大于添加个别代理人进行此项工作的代价，所以相对而言，在此情况下添加代理人反而降低了消耗成本。

对于 API 与控制器交互过程中的安全问题，由于该 API 直接添加到中间盒当中，当中间盒与控制平面中的控制器交涉过程中与控制平面想联通，与其他平面相隔离，而且跟军 SDN 的特性，这种隔离是一种比较清晰的平面隔离，其次，在 SDN 中控制器与上层应用平面或是南向的数据层面进行数据交涉的过程中，都存在相应都安全策略，交涉前存在认证账户，交互中存在 IPS 等安全措施，在 API 与控制器等交涉过程中这些安全措施都可以为控制器等上述行为提供安全防护防护服务。

2.3　软件代理

如上所述，软件代理可以用作中间盒 API 的替代品。它是一个可选的组件，旨在简化策略执行的采用机制，特别是在中间代码专有或者过于复杂且不能被修改的情况下。代理以透明方式代理控制器和中间盒之间的请求，为了获取/设置策略，代理用来实现由中间盒供应商提供的预先存在的 API 或者基于中间盒存储的数据获取状态信息的独立 API。

2.4　中间盒的状态和数据存储

中间盒有多种类型，每种都有自己的方式来表示这些设备的内部状态。但所有的中间盒都有一个共同点：检查和操纵网络流量。因此，一般通过在流量中执行操作来表示其状态。本文采用类似于[5]中提出的在控制器内部存储关于流量修改信息的方法，这样相对使用外部存储可以使应用程序更快地进行访问[12]。

3　实验验证与结果分析

3.1　实验设置

为了验证策略执行体系结构，在虚拟机（VM）中使用 Mininet2.2.1 软件实现原型。为了验证，选择了两个中间盒来覆盖一些最常见的生产环境：防火墙（IPt-ables[13]）和 IPS（Snort[14]），都使用通用的开源软件。选择 Python 开发软件代理，在每个中间盒中获取和设置策略。

在控制器上运行的应用程序将接收包输入消息，并查找预定义的信息，如 IP 地址，TCP/UDP 端口和应用程序层协议。这些信息用于在中间盒中创建和设置适当的策略。一旦应用程序收到确认策略已成功设置，控制器将更新流规则，以允许通过网络设备将流量转发到中间盒。算法 1 是设置防火墙策略应用程序的一个示例，如下所示。它可以允许 VoIP（voice over Internet protocol）通信通过防火墙使用带有实时传输协议(real-time transport protocol,RTP)和会话启动协议（session initiation protocol,SIP）协议的动态端口。

Ryu 控制器具有 Web 服务器网关接口功能，可以部署具象状态传输 API。原型使用本地 Ryu 函数通过将 Python 函数映射到特定的统一资源位置来实现其 API。

策略执行机制的原型在一组仿真中用两种类型的中间盒（即 IPS 和防火墙）进行评估。此外，为了验证实验结果，本文进行了假设检验。评估的目的是验证提出的机制是否可以在不影响网络和应用性能的情况下实现。预计该原型能够动态修改防火墙策略并从 IPS 接收策略，而不会影响相关的性能指标。这些实验使用在增强的简单因特网协议(simple Internet protocol plus,SIPp)上运行的 VoIP 通信。

Algorithm 1　Algorithm used for policy configuration on a firewall.

```
while application is running do
    receive the Packet-in
    if There is no policy with such information then
        for all Protocol header in the packet-in do
            if Package from link layer then
                Create a flow rule
            else if Packet from the network layer then
                Extract source and destination addresses
                Update the flow rule
            else if Packet from transport layer then
                Extract destination port
                Update the flow rule
                if UDP packet then
```

```
            Set the flow rule as a UDP flow
        else if RTP or SIP packet then
            Create a policy with information extracted from the packet-in
            Send the policy to the firewall
            if Policy successfully configured then
                Update the flow rule to forward the data to its desti-nation
        Send the flow rule to the switch
    else
        bypass
```

评估考虑了两种情况：动态策略的一种情况，它实现了本文提出的体系结构，另一种情况是静态策略，它由静态中间盒策略组成。所提出架构造成的网络影响是通过在 SIPp 输出中显示的附加延迟或响应时间造成的，以 Wireshark[15]获得的抖动和分组丢失来衡量。上述性能指标如下所述：

a)延迟或响应时间：网络完全响应应用程序请求需要多长时间。对于 VoIP 呼叫来说，延迟为从源到目的地传输分组所造成的延迟。

b)数据包丢失：丢失的数据包数量超过应用程序交换的数据包总数。数据包丢失导致用户体验下降，特别是对于实时和交互式应用程序；

c)抖动：延迟网络中数据包传送的变化。

评估中防火墙和 IPS 操纵 VoIP 应用的流量，防火墙设置由原型动态调整。在产生实验结果之后，进行统计评估以确定架构对网络性能的影响程度。使用的拓扑结构如图 2 所示。创建虚拟接口之后，Mininet 认为本地机器是内部主机，允许本地机器与仿真网络中的主机进行通信。该接口连接到图 2 所示的拓扑结构中的开关 S2。以下是在实验过程中每个 VoIP 呼叫的原型所遵循的步骤：

a)IDS / IPS 在数据包的有效载荷中搜索 SIP 和 RTP 模式。当 IDS / IPS 发现这些模式时，它发送源和目的地 IP 和端口以及应用于流量的动作应用于控制器之上；

b）流量到达与防火墙相连的交换机，向控制器查询应用于此流量的操作；

c)负责防火墙配置的控制器上面的一个应用程序不断地在从交换机收到的包输入消息中寻找 SIP 和 RTP。当找到这些协议时，此应用程序将提取源 IP 地址，目标 IP 地址，目标端口和传输层协议。该信息以 JSON 格式发送到管理防火墙的代理软件，如 src：192.168.2.101，dst：192.168.4.200，dst port：10934，protocol：udp。要将此数据发送给代理，应用程序向 http：//192.168.3.2：500/InsertPolicyFW 地址发送请求。192.168.3.2 地址是防火墙，5000 是 Flask 侦听 REST 请求的端口；

d）软件代理使用从控制器收到的这些信息，使用 python-iptables 库获取在防火墙中设置策略。然后，该代理程序在策略配置正确时响应控制器成功的状态码，或在策略配置失败时响应失败状态；

e）管理防火墙的控制器上的应用程序接收到代理的响应，并设置交换机将流量转发到防火墙；

f）防火墙接收流量并应用控制器之前配置的策略。

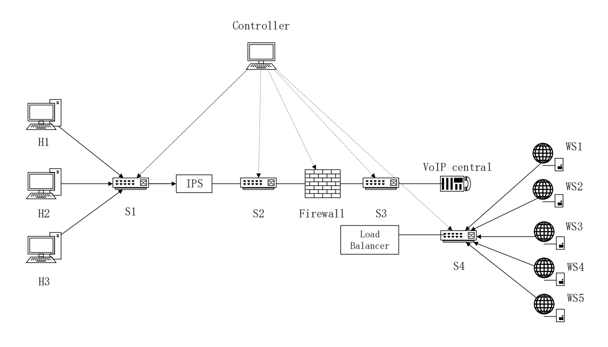


图 2　实验使用的拓扑

Fig.2　Topology used in the experiment

Mininet 是一个广泛使用的平台，用于实验和原型化 SDN 网络[6,7,16-18]。采用比较常用的 VoIP SIPp 流量生成器和 Asterisk 软件[19-22]。根据相关的参考文献[6,7,23,24]选定环境以及中间盒和 VoIP 呼叫的数量。

实验使用 Asterisk 软件[25]作为 VoIP 电话中心，Iptables 防火墙来处理客户端和中心之间的所有流量，Snort IPS 分析网络流量并识别流向 VoIP 中心的流量，Ryu 控制器执行策略执行应用程序，以及启动 VoIP 呼叫的客户端主机。所有实验都使用 RTP 协议进行数据传输，并使用 SIP 协议进行信号传输。IPS 根据有效载荷分析识别 VoIP 呼叫。IPS 有两个规则来识别 VoIP 流量，一个用于 SIP 数据包，另一个用于 RTP 协议。在实验中，Snort IPS 被配置为使用其关键字“content”在分组的有效载荷中搜索 SIP 和 RTP 模式。

在实验开始之前，防火墙被配置为阻止任何超过 1023 的目标端口的连接，所有其他端口被阻塞。但是，这种配置通常会导致问题出现，因为 VoIP 通信可能使用动态高端端口进行通信。本文的策略执行原型应该能够被动地配置防火墙而不需要手动干预以减轻这个问题。

在实验过程中，原型发送和接收策略。但是，从 IPS 接收的策略未用于防火墙配置。防火墙配置基于控制器收到的包输入消息进行了详细描述。

发送到中间盒的设置是由控制器根据网络流量创建的。在本例中，发送到防火墙的策略是基于包消息获得的信息创建的。表 1 显示了实验期间 SDN 控制器在防火墙中配置的策略示例。表 2 显示了 IPS 在实施过程中应用的规则。在此表中，关键字“any”表示任何地址或端口，“content”列表示 IPS 在数据包中搜索的模式。

图 2 中的拓扑在 Mininet 上与使用 SIPp 的客户端主机仿真以生成 50 个同时呼叫。3RTP 流量由 SIPp 中预先加载的默认捕获文件生成。当 SIPp 执行结束时，会产生统计信息，如呼叫总数，呼叫成功次数及响应次数，呼叫失败次数。此次实验中共产生了 20000 个呼叫。

表 1　实验期间由原型配置的防火墙策略示例

Table 1　Example of a firewall policy configured by the prototype during the experiment

动作	协议	源地址	目的地址	目的端口号
接受	UDP	192.168.2.101	192.168.4.200	11280(RIP)
接受	UDP	192.168.2.101	192.168.4.200	14168(RIP)
接受	UDP	192.168.2.101	192.168.4.200	19406(RIP)
接受	UDP	192.168.4.200	192.168.2.101	5060(SIP)
接受	UDP	192.168.2.101	192.168.4.200	5060(SIP)
下降	UDP	anywhere	anywhere	1024:65535

表 2　实验过程中由 Snort 使用的规则

Table 2　Rules used by Snort during the experiment

动作	协议	源地址	源端口	目的地址	目的端口	内容
警报	UDP	any	any	any	any	RTP
警报	UDP	any	any	any	any	SIP

3.2　结果分析比较

在防火墙和 IPS 的实验中，所有的 VoIP 呼叫都是成功的，这表明所提出的策略执行架构已经成功地根据需要操纵防火墙规则。原型也成功地获得了 IPS 政策。在所有情况下都没有数据包丢失，表 3 显示了静态策略和动态策略方案的 VoIP 呼叫的响应时间，表 4 显示了抖动的比较。可以看出呼叫时延和抖动差异都较小，表明有和没有提出的架构的性能结果是非常相似的。

表 3　使用防火墙的 VoIP 呼叫时延

Table 3　VoIP call delay using firewall

时延	平均值	中值	最小值	最大值
动态策略（ms）	5.577	4.460	1.076	78.58
静态策略（ms）	6.465	5.236	1.128	130.0
差异（ms）	-0.888	-0.776	—	—

表 4　使用防火墙和 IPS 的 VoIP 抖动

Table 4　VoIP jitter using firewall and IPS

时延	平均值	中值	最小值	最大值
动态策略（ms）	0.541	0.490	0.300	5.17
静态策略（ms）	0.5876	0.540	0.030	4.33
差异（ms）	-0.0466	-0.05	—	—

为了确认使用原型的性能没有显着变化的假设（即原型的结果在统计上等于静态政策的结果，置信水平为 95%），本文利用抖动和响应时间结果进行了 Kolmogorov-Smirnov（KS）依从性测试以检查数据是否遵循正态分布，用 R 软件计算 p 值，其值小于 0.05，意味着数据不遵循正态分布。因此，本文需要考虑中位数而不是平均值的非参数检验。

本文选择的非参数检验是 Wilcoxon 签名秩检验（WSRT）。本文对配对数据进行了测试，所有数据来自同一网络，动态策略场景代表的实验组，静态策略场景代表的控制组。测试考虑了两个样本的调用次数 n 等于 20000，　。假设是：

H_0:原型不影响性能指标(抖动和响应时间)

H_a:原型影响性能指标(抖动和响应时间)

配对样本的 WSRT 导致 p 值为 2.2^{-16} 为响应时间度量。由于 $2.2^{-16} < 0.05$，本文拒绝了 H_0，这导致中位数在统计上不同的结论。但是，原型（动态策略）的场景比静态策略的响应时间更短。因此，进行第二次 WSRT 测试，以评估动态策略方案是否确实表现出了更好的性能。定义假设：

$$H_0 :\sim \mu_0 > \mu_1$$
$$H_1 :\sim \mu_0 <\sim \mu_1$$

其中 $\sim \mu_0$ 是静态策略样本的中位数。在这个测试中，p 值是 1，所以 H_0 没有被拒绝。因此，动态原型执行策略的响应时间的中位数在统计上低于静态策略的中位数。换句话说，统计上原型不会对 VoIP 呼叫的响应时间造成负面影响。

对于抖动指标，WSRT 测试返回 p 值为 2.2^{-16}，这导致 H_0 的拒绝。正如在响应时间分析中，有如下假设：

$$H_0 :\sim \mu_0 >\sim \mu_1$$
$$H_1 :\sim \mu_0 <\sim \mu_1$$

从这个测试得到的 p 值是 1，H_0 不排斥。因此，原型对抖动没有负面影响。综上可以说明，所提出的体系结构的实现没有给原网络增加负担。

4　结束语

本文提出了一种新的架构来动态执行 SDN 网络中的中间盒策略。尽管中间盒对于网络运行是必不可少的，但传统上它们是由管理员以手动和静态的方式配置的，这使得它们容易出错并且不够灵活，无法处理现代应用。在所提出的架构中，SDN 被用来自动配置最优中间盒策略。所提出的架构在 Mininet 仿真环境中实现为运行的原型。在两个广泛使用的中间盒（防火墙和 IPS）的实验中对原型进行了评估，使用不同的网络尺寸来评估其可扩展性。实验考虑了 VoIP 应用，并分析了相关的性能指标（响应时间，数据包丢失和抖动）。原型能够自动和动态地强制执行中间盒策略。此外，原型在所有评估的情况下都不会影响网络的性能。因此，本文介绍的架构被证明是在 SDN 网络中配置中间盒策略的有效选择。它提出了一种新的使用集权的方式可编程性在 SDN 体系结构中呈现，使中间盒配置更容易和更有效。

参考文献：

[1] Carpenter B, Brim S. RFC3234, Middleboxes: taxonomy and issues [S]. 2002.

[2] Stiemerling M, Quittek J, Cadar C. RFC4540, NEC's simple middlebox configuration SIMCO) protocol version 3. 0 [S]. 2006.

[3] Katti S, Levis P, McKeown N, *et al.* Software-defined networking [EB/OL]. https://ee. stanford. edu/research/software-defined-networking.

[4] Gemberjacobson A, Viswanathan R, Prakash C, *et al.* OpenNF: enabling innovation in network function control [J]. ACM SIGCOMM Computer Communication Review, 2014, 44 (4): 163-174.



[5] Gember A, Prabhu P, Ghadiyali Z, *et al.* Toward software-defined middlebox networking [C]// Proc of ACM Workshop on Hot Topics in Networks. New York: ACM Press, 2012: 7-12.

[6] Fayazbakhsh S K, Chiang L, Sekar V, *et al.* Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags [C]// Proc of USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2014: 533-546.

[7] Qazi Z A, Tu C C, Chiang L, *et al.* SIMPLE-fying middlebox policy enforcement using SDN [J]. ACM SIGCOMM Computer Communication Review, 2013, 43 (4): 27-38.

[8] Cao Zizhong, Kodialam M, Lakshman T V. Traffic steering in software defined networks: planning and online routing [J]. ACM SIGCOMM Computer Communication Review, 2014, 44 (4): 65-70.

[9] Xuan Leifei, Wu Peifei. The Optimization and implementation of iptables rules set on Linux [C]// Proc of International Conference on Information Science and Control Engineering. 2015: 988-991.

[10] Alhomoud A, Munir R, Disso J P, *et al.* Performance evaluation study of intrusion detection systems [J]. Procedia Computer Science, 2011, 5 (9): 173-180.

[11] RYU project team. Ryu SDN framework [EB/OL]. [2018-05-05]. https://osrg. github. io/ryu.

[12] Krishnamurthy A, Chandrabose S P, Gember-Jacobson A. Pratyaastha: an efficient elastic distributed SDN control plane [C]// Proc of Workshop on Hot Topics in Software Defined Networking. New York: ACM Press, 2014: 133-138.

[13] Netfilter Core Team. The netfilter. org“iptables”project [EB/OL]. [2018-05-05]. http://www. netfilter. org/projects/iptables/.

[14] Snort Core Team. Snort: intrustion detection/prevention system [EB/OL]. (2016) [2018-05-05]. https://www. snort. org/.

[15] Orebaugh A, Ramirez G, Burke J, *et al.* Wireshark & Ethereal network protocol analyzer toolkit (Jay Beale's open source security) [C]// Syngress Publishing, 2006: 523–540.

[16] Atary A, Bremler-Barr A. Efficient round-trip time monitoring in OpenFlow networks [C]// Proc of IEEE International Conference on Computer Communications. Piscataway, NJ: IEEE Press, 2016: 1-9.

[17] Espinet F, Joumblatt D, Rossi D. Framework, models and controlled experiments for network troubleshooting [J]. Computer Networks, 2016, 107: 36-54.

[18] Wang Bing, Zheng Yao, Lou Wenjing, *et al.* DDoS attack protection in the era of cloud computing and Software-Defined Networking [J]. Computer Networks, 2015, 81 (C): 308-319.

[19] Femminella M, Francescangeli R, Giacinti F, *et al.* Design, implementation, and performance evaluation of an advanced SIP-based call control for VoIP services [C]// Proc of IEEE International Conference on Communications. . Piscataway, NJ: IEEE Press, 2009: 1465-1469.

[20] Rafique M Z, Akbar M A, Farooq M. Evaluating DoS Attacks against Sip-Based VoIP Systems [C]// Proc of Global Telecommunications Conference. . Piscataway, NJ: IEEE Press, 2010: 1-6.

[21] Chamorro V G, Castillo C N, Lopez-Pires F. An Elastic VoIP Solution Based on OpenStack [C]// Proc of International Conference on Information Systems Engineering. . Piscataway, NJ: IEEE Press, 2016: 43-47.

[22] Stanek J, Kencl L. SIPp-DD: SIP DDoS flood-attack simulation tool [C]// Proc of International Conference on Computer Communications and Networks. . Piscataway, NJ: IEEE Press, 2011: 1-7.

[23] Yang H Y, Lee K H, Ko S J. Communication quality of voice over TCP used for firewall traversal [C]// Proc of IEEE International Conference on Multimedia and Expo. . Piscataway, NJ: IEEE Press, 2008: 29-32.

[24] Zhou Dawen, Huang Benxiong, Mo Yijun. Distributed architecture of VOIP for firewall//NAT Traversing [C]// Proc of International Conference on Wireless Communications, Networking and Mobile Computing. . Piscataway, NJ: IEEE Press, 2005: 1160-1163.

[25] Imran A, Qadeer M A, Khan M J R. Asterisk VoIP private branch exchange [C]// Proc of International Conference on Multimedia, Signal Processing and Communication Technologie. . Piscataway, NJ: IEEE Press, 2009: 217-220